

関数型プログラミング

第3回 基本(2) 型と高階関数

萩野 達也

hagino@sfc.keio.ac.jp

Slide URL

<https://vu5.sfc.keio.ac.jp/slide/>

型と値

- 値は型ごとに分類されている
 - 型は値の集合
- Haskellは静的な型チェックを行う
 - コンパイル時に型をチェックしてくれる
 - 型が合わないとエラーになる
- Haskellは型推論を行う
 - 明示的に型を指定しなくとも推論して補ってくれる

基本的な型

型	意味	値の例
<code>Int</code>	整数値	<code>-1, 0, 1, 777</code>
<code>Char</code>	文字	<code>'a', 'A', '\n'</code>
<code>String</code>	文字列	<code>"abc", "string"</code>
<code>Bool</code>	真偽値	<code>True, False</code>
<code>[OO]</code>	OOのリスト	<code>[1, 2, 3]</code>

- 真偽値

- 真が`True`, 偽が`False`

- リスト

- `[Int]` 整数のリスト型
- `[Char]` 文字のリスト型 (別名は`String`型)

関数の型

第1引数の型 -> 第2引数の型 -> …… -> 返り値の型

- `lines`の型
 - `String -> [String]`
- `unlines`の型
 - `[String] -> String`
- `firstNLines n cs = unlines $ take n $ lines cs`
 - `Int -> String -> String`

型変数

- **length**関数
 - `length [1, 2, 3]`
 - `length ['a', 'b']`
 - `length ["abc", "def"]`
 - 色々な型のリストに適用可能
 - 多層(polymorphic)関数
 - **length関数の型**
 - `[a] -> Int`
 - `a`は型変数
- **take**関数
 - `take 3 [1, 2, 3]`
 - **take関数の型**
 - `Int -> [a] -> [a]`

ghciを使って型の調べ方

```
Prelude> :type take
take :: Int -> [a] -> [a]
```

型の宣言

変数名 :: 型

- 変数の型を指定する

関数名 :: 第1引数の型 -> 第2引数の型 -> …… -> 戻り値の型

- 関数の型を指定する
 - 関数の定義の時にチェックされる
 - 型推論がうまくいかないときに指定する

```
length  :: [a] -> Int
reverse :: [a] -> [a]
take    :: Int -> [a] -> [a]
words   :: String -> [String]
lines   :: String -> [String]
unlines :: [String] -> String
print   :: (Show a) => a -> IO()
putStr  :: String -> IO()
putStrLn :: String -> IO()
```

高階関数

- 関数も値
 - 整数値などと同じように関数も値として扱うことができる
- 高階関数
 - 関数の値を引数として取る関数
 - 関数を値として返す関数
- `map`関数
 - `map :: (a -> b) -> [a] -> [b]`
 - `map square [1, 2, 3]`
 - `[(square 1), (square 2), (square 3)]`
 - `[1, 4, 9]`
 - ここで `square n = n * n`
 - `map f xs`
 - リスト`xs`の各要素に`f`を適用したリストを返す

expandコマンドの作成

```
expand.hs
```

```
main = do cs <- getContents
         putStr $ expand cs

expand :: String -> String
expand cs = map translate cs

translate :: Char -> Char
translate c = if c == '\t' then '@' else c
```

- 上記のプログラムを入力して、実行してみなさい。
 - タブを@に置き換えるコマンド

```
% ghc expand.hs
...
% ./expand < airline-code.txt
Q5@40-Mile Air@MILE-AIR@United States
W9@Abelag Aviation@ABG@Belgium
...
```


if式

```
if 条件式 then 式1 else 式2
```

- C言語やjavaのif文とは少し違います
 - 3項演算子の「式1?式2:式3」と同じ
- if式の値
 - 条件式の値がTrueならば式1の値になり, Falseならば式2の値になる

```
translate c = if c == '\t' then '@' else c
```

- `c`がタブだったら '@' で, そうでない場合には `c`
 - `translate '\t' → '@'`
 - `translate 'a' → 'a'`
 - `translate '\n' → '\n'`

expandの説明

```
expand :: String -> String
expand cs = map translate cs
```

- `translate`はタブ文字を@に変換する関数
 - `translate :: Char -> Char`
- `map translate cs`
 - `cs`は文字列(`Char`の配列)
 - `map`によって`cs`の各文字に`translate`を適用した結果の文字列になる
 - `map translate "abc\tdef\n"`
 - `map translate ['a', 'b', 'c', '\t', 'd', 'e', 'f', '\n']`
 - `[(translate 'a'), (translate 'b'), (translate 'c'), (translate '\t'), (translate 'd'), (translate 'e'), (translate 'f'), (translate '\n')]`
 - `['a', 'b', 'c', '@', 'd', 'e', 'f', '\n']`
 - `"abc@def\n"`

==演算子

- == は二項演算子
- (==) が関数名
 - (==) :: a -> a -> Bool
 - `x == y`
 - `x`と`y`が等しいときに`True`を返し
 - 等しくないときに`False`を返す

if式の書き方

```
if c == '\t' then '@' else c
```

- thenやelseの後の式が大きくなった場合には改行して書いてもかまわない。

```
if c == '\t'  
  then '@'  
  else c
```

- あるいは そろえる

```
if c == '\t' then '@'  
           else c
```

練習問題3-1

- 各行の先頭に'\t'を追加するコマンドtab.hsを完成させなさい。

```
tab.hs
```

```
main = do cs <- getContents
          putStr $ unlines $ map addTab $ lines cs

addTab :: String -> String
addTab cs = ...
```

- `addTab cs`は`cs`の先頭に'\t'を追加する関数です。
 - リストを連結するには`++`演算子を使うことができます。
 - `[1, 2, 3] ++ [4, 5] → [1, 2, 3, 4, 5]`
 - `"abc" ++ "def" → "abcdef"`

```
% ghc tab.hs
...
% ./tab < airline-code.txt
      Q5      40-Mile Air      MILE-AIR United States
      W9      Abelag Aviation  ABG      Belgium
      M3      ABSA Cargo      Turismo  Brazil
...
```

expand(バージョン2)コマンドの作成

```
expand2.hs
```

```
main = do cs <- getContents
         putStr $ expand cs
```

```
expand :: String -> String
expand cs = concat $ map expandTab cs
```

```
expandTab :: Char -> String
expandTab c = if c == '\t' then "      " else [c]
```

- 最初のexpandではタブが1文字の@に置き換えられた
 - タブは8文字の空白と置き換えたい
 - 上記のプログラムを入力して実行してみなさい。

```
% ghc expand2.hs
...
% ./expand2 < airline-code.txt
Q5      40-Mile Air      MILE-AIR      United States
W9      Abelag Aviation  ABG           Belgium
...
```

concat

```
concat :: [[a]] -> [a]
```

- リストのリストが与えられたときに, それらのリストを連結して一つのリストにする.
 - 2重のリストを1重にする (flat化)
 - `concat [[1, 2], [3], [4, 5]] → [1, 2, 3, 4, 5]`
 - `concat [[1, 2], [], [3]] → [1, 2, 3]`
 - `concat ["ab", "c", "de"] → "abcde"`
 - `concat ["ab", "", "cd"] → "abcd"`

expand(バージョン3)コマンドの作成

```
expand3.hs
```

```
tabStop = 8
```

```
main = do cs <- getContents
         putStr $ expand cs
```

```
expand :: String -> String
expand cs = concatMap expandTab cs
```

```
expandTab :: Char -> String
expandTab '\t' = replicate tabStop ' '
expandTab c   = [c]
```

- expandTabをパターンマッチを使って書き直す.
- concatとmapを組み合わせたconcatMapを利用する.

```
% ghc expand3.hs
```

```
...
```

```
% ./expand3 < airline-code.txt
```

```
Q5          40-Mile Air          MILE-AIR          United States
```

```
W9          Abelag Aviation      ABG               Belgium
```

```
...
```


パターンマッチによる関数の定義

```

expandTab :: Char -> String
expandTab '\t' = replicate tabStop ' '
expandTab c    = [c]

```

- 引数の値のパターンに応じて場合分けして書くことができる。
 - 引数の値が'\t'の時には`replicate tabStop ' '`となる
 - それ以外の場合には`[c]`となる

関数名	第1引数のパターン	第2引数のパターン	……	=	定義1
関数名	第1引数のパターン	第2引数のパターン	……	=	定義2
関数名	第1引数のパターン	第2引数のパターン	……	=	定義3
:	:	:		:	:
:	:	:		:	:

concatMap関数とreplicate関数

- **concatMap関数**

- `concatMap :: (a -> [b]) -> [a] -> [b]`
- `concat`と`map`をまとめた関数
- `concatMap f xs = concat $ map f xs`

- **replicate関数**

- `replicate :: Int -> a -> [a]`
- `replicate n x = x`を`n`個含むリストを返す
- `replicate 3 True → [True, True, True]`
- `replicate 3 77 → [77, 77, 77]`
- `replicate 3 'a' → "aaa"`
- `replicate 0 True → []`

練習問題3-2

```
lower.hs
```

```
main = do cs <- getContents
         putStr $ map lower cs

lower :: Char -> Char
lower 'A' = 'a'
...
lower c = ...
```

- ファイルの大文字をすべて小文字にするlowerコマンドを完成させなさい.

```
% ghc lower.hs
...
% ./lower < airline-code.txt
q5      40-mile air      mile-air united states
w9      abelag aviation   abg      belgium
m3      absa cargo        turismo  brazil
...
```

練習問題3-3

```
sfc.hs
```

```
main = do cs <- getContents
         putStr ...
```

- ファイルの「S」を「Shonan」に, 「F」を「Fujisawa」に, 「C」を「Campus」に置き換えて出力するsfcコマンドを完成させなさい.
 - 'S' → "Shonan "
 - 'F' → "Fujisawa "
 - 'C' → "Campus "

```
% ghc sfc.hs
```

```
...
```

```
% ./sfc
```

```
SFC
```

```
Shonan Fujisawa Campus
```

```
Super Car
```

```
Shonan upper Campus ar
```

```
...
```

入力

出力

関数のまとめ

関数名	適用例	意味
<code>map</code>	<code>map f xs</code>	リスト <code>xs</code> の各要素に関数 <code>f</code> を適用したリストを返す
<code>concat</code>	<code>concat xs</code>	リストのリスト <code>xs</code> の要素を連結して一重のリストにする
<code>concatMap</code>	<code>concatMap f xs</code>	<code>concat (map f xs)</code> と同じ
<code>replicate</code>	<code>replicate n x</code>	<code>x</code> を <code>n</code> 個含むリストを返す
<code>(==)</code>	<code>x == y</code>	<code>x</code> と <code>y</code> が等しいときに <code>True</code> を返す
<code>(++)</code>	<code>xs ++ ys</code>	リスト <code>xs</code> とリスト <code>ys</code> を連結する